Error detection/correction

(Maintaining data (information) integrity across noisy channels and less-than-reliable storage media)

**Error detection** is the detection of errors caused by noise or other impairments during transmission from the transmitter to the receiver.

**Error correction** is the detection of errors and reconstruction of the original, error-free data.

Error correction may generally be realized in two different ways:
1) Automatic repeat request (ARQ) (sometimes also referred to as backward error correction)
2) Forward error correction (FEC)
Backwards and Forwards error correction

**Automatic repeat request (ARQ)** (sometimes also referred to as backward error correction): This is an error control technique whereby an error detection scheme is combined with requests for retransmission of erroneous data. Every block of data received is checked using the error detection code used, and if the check fails, retransmission of the data is requested – this may be done repeatedly, until the data can be verified.

**Forward error correction (FEC):** The sender encodes the data using an error-correcting code (ECC) prior to transmission. The additional information (redundancy) added by the code is used by the receiver to recover the original data. In general, the reconstructed data is what is deemed the "most likely" original data.

Forward error correction (FEC) code is a system of adding redundant data, or parity data, to a message, such that it can be recovered by a receiver even when a number of errors (up to the capability of the code being used) were introduced, either during the process of transmission, or on storage.
Codes - General

- A code that will detect $2t$ or fewer errors can correct $t$ or fewer errors.

- A code will detect all sets of $t$ or fewer errors iff the minimum hamming distance between code words is at least $t + 1$.
  \[ t + 1 = 3; \text{detects 2 errors} \]

- A code is capable of correcting all sets of $t$ or fewer errors iff the minimum hamming distance between code words is at least $2t + 1$.
  \[ 2t + 1 = 3; \text{corrects 1 error} \]
Codeword polynomial:

\[ V(x) = R(x) + X_{n-k} M(x) \]

where:

- \( V(x) \) = codeword;
- \( R(x) \) = remainder poly.;
- \( X_{n-k} \) = shift bits;
- \( M(x) \) = message bits
**Codeword generation** example:

Encode the message 101 as a (7,3) code with a generator polynomial of:

\[ X^4 + X^3 + X^2 + 1. \]
Codeword generation example:

Message = 1 0 1

\[ M(x) = 1 + 0x + x^2 \]

\[ X^{n-k} M(x) = x^4 M(x) = 0 + 0x + 0x^2 + 0x^3 + x^4 + 0x^5 + x^6 \]

\[ R(x) = 1 + x \quad \text{<--see next slide} \]

\[ V(x) = 1 + x + 0x^2 + 0x^3 + x^4 + 0x^5 + x^6 \]

Codeword = 1 1 0 0 1 0 1

Check Digits  |  Message
R(x) calculation:

\[
\begin{array}{c}
X^4 + X^3 + X^2 + 0 + 1 \\
| X^6 + 0X^5 + X^4 \\
X^6 + X^5 + X^4 + 0X^3 + X^2 \\
\hline
X^5 + X^4 + X^3 + X^2 + X \\
X^4 + X^3 + X^2 + X \\
X^4 + X^3 + X^2 + 1 \\
\hline
X + 1
\end{array}
\]

XOR process
**CRC** - Cyclic redundancy check polynomials

CRC-5  = $X^5 + X^4 + X^2 + X$

CRC-16 = $X^{16} + X^{15} + X^5 + 1$

CRC-32  [Used by ethernet, FDDI, ZIP, PNG and others]

\[ C(x) = x^{31} + x^{30} + x^{26} + x^{25} + x^{24} + x^{18} + x^{15} + x^{14} + x^{12} + x^{11} + x^{10} + x^8 + x^6 + x^5 + x^4 + x^3 + x + 1 \]
**CRC - Cyclic redundancy check polynomials**

**Designing CRC polynomials**
The selection of generator polynomial is the most important part of implementing the CRC algorithm. The polynomial must be chosen to maximize the error detecting capabilities while minimizing overall collision probabilities. The most important attribute of the polynomial is its length (the number of the highest nonzero coefficient), because of its direct influence of the length of the computed checksum.

The most commonly used polynomial lengths are
- 9 bits (CRC-8)
- 17 bits (CRC-16)
- 33 bits (CRC-32)
- 65 bits (CRC-64)

A polynomial $g(x)$ that admits other factorizations may be chosen then so as to balance the maximal total blocklength with a desired error detection power. The BCH codes are a powerful class of such polynomials.
# CRC - Cyclic redundancy check polynomials

<table>
<thead>
<tr>
<th>Name</th>
<th>Uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC-1</td>
<td>most hardware; also known as parity bit</td>
</tr>
<tr>
<td>CRC-4-ITU</td>
<td>G.704</td>
</tr>
<tr>
<td>CRC-5-ITU</td>
<td>Gen 2 RFID[^{15}]</td>
</tr>
<tr>
<td>CRC-5-USB</td>
<td>USB token packets</td>
</tr>
<tr>
<td>CRC-7</td>
<td>telecom systems, G.707, G.832, MMC, SD</td>
</tr>
<tr>
<td>CRC-8-CCITT</td>
<td>I.432.1; ATM HEC, ISDN HEC and cell delineation</td>
</tr>
<tr>
<td>CRC-8-Maxim</td>
<td>1-Wire bus</td>
</tr>
<tr>
<td>CRC-8-SAE</td>
<td>AES3</td>
</tr>
<tr>
<td>J1850</td>
<td>ATM; I.610</td>
</tr>
<tr>
<td>CRC-10</td>
<td>telecom systems</td>
</tr>
<tr>
<td>CRC-12</td>
<td>Bisync, Modbus, USB, ANSI X3.28, SIA DC-07, many others; also known</td>
</tr>
<tr>
<td>CRC-16-IBM</td>
<td>as CRC-16 and CRC-16-ANSI</td>
</tr>
<tr>
<td>CRC-16-CCITT</td>
<td>X.25, V.41, HDLC FCS, XMODEM, Bluetooth, PACTOR, SD, many others;</td>
</tr>
<tr>
<td>CRC-16-T10-DIF</td>
<td>known as CRC-CCITT</td>
</tr>
<tr>
<td>CRC-16-DECT</td>
<td>cordless telephones[23]</td>
</tr>
<tr>
<td>CRC-17-CAN</td>
<td>CAN FD</td>
</tr>
<tr>
<td>CRC-21-CAN</td>
<td>CAN FD</td>
</tr>
<tr>
<td>CRC-24</td>
<td>FlexRay</td>
</tr>
<tr>
<td>CRC-24-Radix-64</td>
<td>OpenPGP, RTCM104v3</td>
</tr>
<tr>
<td>CRC-30</td>
<td>CDMA</td>
</tr>
<tr>
<td>CRC-32</td>
<td>HDLC, ANSI X3.66, ITU-T V.42, Ethernet, Serial</td>
</tr>
<tr>
<td>CRC-40-GSM</td>
<td>GSM control channel</td>
</tr>
<tr>
<td>CRC-64-ISO</td>
<td>HDLC, Swiss-Prot/TrEMBL; considered weak for hashing</td>
</tr>
<tr>
<td>ECMA-182</td>
<td>ECMA-182, XZ Utils</td>
</tr>
</tbody>
</table>